

NOVEMBRE 1986

PASCAL
PROLOG
FORTH
LISP
APL

EDITORIAL

Ouf, nous sommes presque en 1987 et, enfin, le FORTH 83-Standard démarre en France. A l'allure où nous écoulons la version MSDOS, notre revue risque de s'orienter vers un public d'utilisateur PC et compatibles. Y-aurait-il comme une préférence pour ce système pourtant tant critiqué à son apparition? Alors n'oubliez pas vos ZAPPEL, ZATARI et cie pour faire le poids face à l'invasion des clones de ZIBEHEM.

Et je m'arrète là, car j'ai la crampe du dactylo...

(S'il y a des volontaires pour faire des éditos marrants, je suis preneur)

SOMMAIRE

ASSOCIATION JEDI 8, rue Poirier de Narçay 75014 PARIS

Tel: (1) 45.42.88.90 (de 10h à 18h)

VARIABLES LOCALES ..... (Suite .)
LES PSEUDO-CONSTANTES

par Dominique GEORGE

Bravo à l'article de PETREMANN sur les variables locales . Il faut rendre FORTH plus relisible . Même à l'écriture la manipulation et surtout l'optimalisation des PICK SWAP DROP ROT est souvent bien trop laborieuse .
Pour ma part il y a une solution assez simple répondant à l'objectif de relisibilité mais ne se prêtant pas au jeu de variables locales .Elle est toutefois à manier avec précaution puisqu' elle ne peut être employée qu'à l'intérieur d'une définition deux points .

Elle consiste dans l'emploi de données que j'appelerai *pseudo -constantes* qui fonctionnent de la façon suivante :

- au départ elles sont déclarées comme des constantes selon les règles habituelles
- en lecture elles restituent leur valeur par simple appel de leur nom tout comme les constantes
- en écriture on change leur valeur à l'aide du mot TO ou mieux -> pour une meilleure relecture . Avec la syntaxe d'emploi suivante :

n -> Nom

où n est la nouvelle valeur déposée au préalable au sommet de la pile et Nom le nom de la pseudo-constante .

Grace à cela nous revenons au symbolisme mathématique courant où le nom d'une variable représente sa valeur alors qu 'en FORTH il ne correspond qu'à l'adresse mémoire de sa valeur .

Sans plus attendre voici la définition théorique du mot ->

```
: -> ( n --  ) R> DUP 2+ >R ● 2+ ! ;
```

Les initiés comprennent très vite que la précaution d'emploi annoncée plus haut provient de la manipulation de la Return Stack qui ne peut " marcher" qu'à l'intérieur d'une définition deux- points . Précaution cependant très facile à retenir et à observer .

Pour bien comprendre le fonctionnement du mot quelques rappels sont nécessaires :

Un CFA est une valeur pointant (ou l'adresse de) la position mémoire où commencent à proprement parler la définition spécifique de chacun des differents mots du dictionnaire Forth .

C'est ainsi que la valeur d'une constante se trouve deux octets après l'adresse pointée par son CFA
Dans le dictionnaire la définition compilée proprement dite du mot -> qui s'écrivait au clavier : -> R> DUP 2+ >R ● 2+ ! ; se présente comme une suite de 9 cellules :
la premiere contenant le CFA de :
la seconde le CFA de R>
la 3eme le CFA de DUP
la 4eme le CFA de 2+
la 5eme le CFA de >R
la 6eme le CFA de ●
la 7eme le CFA de 2+
la 8eme le CFA de !
la 9eme le CFA de ;
Par ailleurs pour fonctionner l'interprête du FORTH s'appuie sur deux règles
Règle 1 :
L'exécution d'un mot A1 présent dans la Nième cellule d'une définition deux-points est toujours précédée par le chargement dans IP de l'adresse de la N+1ième cellule .
Règle 2 :
Lorsque ce mot A1 lui même obéit à une définition deux-points c'est à dire s 'écrit au clavier sous la forme : A1 B1 B2 B3 B4 ;
il y a , avant execution de B1, sauvegarde préalable sur la Return Stack du contenu de IP
Le : déclenche cette sauvegarde tandis que la restitution est déclenchée par le ; qui fait suite à B4

Revenons à notre mot -> et considérons l'exemple suivant :

```
5 CONSTANT  COUNT
: OCOUNT  0 -> COUNT ;
```

le corps de la définition de OCOUNT dans le dictionnaire comprend 5 cellules
la 1ere contient le CFA de :
la 2eme le CFA de 0 (considéré comme une constante)
la 3eme le CFA de ->
la 4eme le CFA de COUNT
la 5eme le CFA de ;

Déclenchons maintenant son exécution

Lorsque 0 est déposé sur la pile IP pointe déja sur la 3eme cellule de OCOUNT (->)

Lorsque l'exécution de -> commence IP a été incrémenté pour pointer sur la 4eme cellule de OCOUNT qui contient le CFA COUNT

Le : sauvegarde dans la Return Stackl'adresse de cette 4eme cellule (regle 2) puis fait pointer IP sur la 2eme cellule de ->

R> transfert dans la pile l'adresse de la 4eme cellule de OCOUNT

Après duplication la séquence 2+ >R renvoie sur la Return Stack non plus l' adresse de la 4eme cellule mais celle de la 5eme

● donne le contenu de l'adresse dupliquée c'est à dire le CFA de COUNT

2+ ! modifie la valeur de la pseudo-constante avec la valeur 0 restée sur la pile

le ; de la fin du mot -> restitue dans IP le contenu de la Retun Stack c'est à dire l'adresse de la 5ieme cellule de OCOUNT et amorce la fin du processus si bien qu'en fin de compte le mot COUNT a été proprement escamoté , ce qui était l'objectif recherché .

Bien qu'assez conçis il est souhaitable d'écrire ce mot en assembleur pour en faire une primitive aussi rapide que son homogue ! qu'il a vocation de concurrencer .

Pour ma part travaillant sur Macintosh la solution est la suivante :

```
CODE -> IP )+ DO WORD MOVE,
     PL IF,  SP )+ DO 2 WORD A4  ●I) LONG MOVE, NEXT THEN,
             SP )+ DO 2 WORD A5  ●I) LONG MOVE, NEXT   END-CODE
```

Commentaires pour les fanas de MacFORTH:

Le mot n'obeit pas à une définition deux -points : il n'y a donc pas lieu de manipuler la Retun Stack .

En préalable à l'exécution IP pointe déja sur la cellule contenant le CFA (à plus proprement parler le token de la pseudo constante)

IP)+ DO WORD MOVE, post incrémente IP ce qui réalise l'escamotage ultérieur de l'exécution de la pseudo variable

Si le contenu de DO c'est à dire le TOKEN de la PSEUDO-CONSTANTE est positif c'est à dire inférieur à 32767 la valeur à modifier est à l'adresse égale à la somme du TOKEN augmenté de 2 et de l'adresse de la base du dictionnaire contenue dans le registre A4 (relogeabilité des blocs oblige)
Sinon le token négatif doit être ajouté à l'adresse contenu dans A5

Une fois compilée cette définition tient en 14 octects dans le dictionnaire : 301B 6R06 299F 2 4ED4 2B9F 2 4ED4

Chapeau pour la puissance d'adressage du 68000 .

Pour ceux qui ne veulent pas charger l'assembleur pour ce seul mot il est possible alors de passer par la définition
HEX

```
create -> -2 allot 301B w, 6B06 w, 299F w, 2 w, 4ED4 w,
2B9F w,  2 w, 4ED4 w,
```

Avant de terminer remarquons qu'il y a intérêt à lui adjoindre l'équivalent du mot +! lui aussi bien utile
Son symbole serait +-> sa syntaxe la même que pour -> et sa définition

```
: +-> ( n -- ) R> DUP 2+ >R ● 2+ +! ;
```

A vous de jouer .



16 Jedi 25.07 09H49
Taper en minuscule en 83-Standard:
- IBM et AMSTRAD (et tous les autres...)

Vous vous melangez le clavier, entre majuscules et minuscules, la dactylo qui sommeille en vous n'a pas encore fait surface, voici un remede de cheval:

```
-1 CAPS !  ( c'est tout, promis juré)
```

et maintenant, que vous tapiez

```
         WORDS
  ou     words
  ou meme  WoRdS
```

votre FORTH Ne VoUs FeRa PlUs La GuEuLe



```
*|**********************************************************************
ADAPTATION DE READ-EVAL-PRINT:
  {methode objet args} se lit (<objet.package>:methode objet args),
  dans ce cas la méthode est résolue statiquement (plus efficace);
  {@methode objet args} se lit (SEND 'methode objet args),
  dans ce cas la méthode sera résolue dynamiquement (plus général).
  Un objet est invariant et s'imprime: {package (champ valeur)…}
  sa représentation interne est ALCYONE: *(package . *[objvals])

CREATION DE NOUVEAUX OBJETS:
  l'objet racine prédéfini, {obj:} (aucun champ), est lié à OBJ
  {NEW objet} retourne une copie de objet;
  {NEW objet 'var} retourne et lie à var une copie de objet;
  {NEW objet 'var '((nom val)…)} lie à var une extension de objet
  avec les nouveaux champs (nom val)… dont les méthodes d'accès
  sont crées automatiquement (dans le package <obj.package>:<var>):
  {nom var} retourne la valeur du champ nom de l'objet lié à var
  {nom var val} donne la valeur val au champ nom de l'objet lié à var

LISTE ET CREATION DES METHODES:
  {METHOD obj} retourne la liste des noms des méthodes locales à obj
  {METHOD obj 'nom} affiche (pretty) la définition de la méthode nom
  {METHOD obj 'nom '(lvar body)} définit une nouvelle méthode de obj
  (le premier argument de la méthode doit être du type de obj)

EXEMPLES:
? {NEW OBJ 'point '((x 10)(y 20))} ; créons un nouvel objet: POINT
= {*:obj:point (x 10) (y 20)}
? point                            ; il est lié à l'atome POINT
= {*:obj:point (x 10) (y 20)}
? {x point}                        ; lisons son champ X
= 10
? {y point 15}                     ; modifions son champ Y
= 15
? {NEW point 'p1}                  ; lions une copie à P1
= {*:obj:point (x 10) (y 15)}
? {x point 5}                      ; modifions le champ X de POINT
= 5
? p1                               ; P1 n'a pas été modifié
= {*:objet:point (x 10) (y 15)}
? ; on peut bien sûr imbriquer, tant dans les définitions…
? {NEW obj 'rect '((topleft {NEW point}) (h 30) (l 50))}
= {*:obj:rect (topleft {*:obj:point (x 5) (y 15)}) (h 30) (l 50)}
? {x {topleft rect} 11}           ; …que dans les appels
= 11
```

```
? {METHOD point 'move '((point dx dy) ; créons une nouvelle méthode
?  {x point (+ dx {x point})}  {@y point (+ dy {y point})}  point))
= *:obj:point:move
? {METHOD point}                    ; elle est dans l'espace de POINT
= (*:obj:point:move *:obj:point:x *:obj:point:y)
? {METHOD point 'move}              ; vérifions l'effet de {@ (SEND_)
(de *:obj:point:move (point dx dy)
    (*:obj:point:x point (+ dx (*:obj:point:x point)))
    (send 'y point (+ dy (*:obj:point:y point)))
    point)
= ()
? {move p1 2 -3}                    ; ça marche !
= {*:obj:point (x 12) (y 12)}

Et voilà: à vous de jouer avec Le_Lisp_Objet !
******************************************************************|*

(DEFVAR *:sys-package:colon 'obj) ; package par défaut

(DEFVAR obj '*(obj . *[])) ; objet racine, aucun champ

(SYNONYMQ :package CAR) ; lecture du package de l'objet

(SYNONYMQ :objvals CDR) ; lecture du vecteur des valeurs

(DE :super (obj) ; pour forcer la recherche à partir du package père
 (TCONS (PACKAGECELL (:package obj)) (:objvals obj)))

; prédicat pour le type Le_Lisp_Objet = *(package . *[objvals])
(DE objp (obj)
 (AND (TCONSP obj)
      (ATOMP   (:package obj))
      (VECTORP (:objvals obj))
      obj))

; adaptation de READ:
;  {methode objet args} se lit (SEND 'methode objet args),
;  dans ce cas la méthode sera résolue statiquement (plus efficace);
;  {@methode objet args} se lit (SEND 'methode objet args),
;  dans ce cas la méthode sera résolue dynamiquement (plus général).
(DMC |{| ()
 (LET ((:dynamic (WHEN (EQ (PEEKCN) */@) (READCH)))
       (:super   (WHEN (EQ (PEEKCN) */^) (READCH)))
       (:forme))
  (WITH ((TYPECH '|}| 'CMACRO))
   (FLET ((|}| ()
            (EXIT |}|
             (SETQ :forme (NREVERSE :forme)) ; (method obj . args)
```

```
         (WHEN :super
               (RPLACA (CDR :forme) (LIST ':super (CADR :forme))))
         (CONS (IF :dynamic 'SEND 'semd)
               (RPLACA :forme (LIST 'QUOTE (CAR :forme))))))
      (UNTILEXIT |}| (NEWL :forme (READ)))))))

; SEMD résoud statiquement une forme de même syntaxe que SEND
;  (SEMD 'methode objet …) donne (<objet.package>:methode objet …)
(DMD semd (:met :obj . :args)
 (LET ((:method (GETFN (:package (EVAL :obj)) (EVAL :met))))
  (IFN :method
   (ERROR 'semd 'ERRUDF (CONS (:package (EVAL :obj)) (EVAL :met)))
   (AND (CONSP :obj) (EQ (CAR :obj) ':super) (SETQ :obj (CADR :obj)))
   (MCONS :method :obj :args))))

; adaptation de EVAL: un objet est invariant
(DE :eval (obj)
 (OR (objp obj)
     (ERROR ':eval "l'argument n'est pas un objet" obj)))

; adaptation de PRIN: un objet s'imprime {package: (champ valeur)…}
(DE :prin (obj)
 (PRIN "{" (:package obj))
 (FOR (numval 0 1 (1- (VLENGTH (:objvals obj))))
      (PRIN " ("
            (VREF (:objvals (SYMEVAL (:package obj))) numval)
            " "
            (VREF (:objvals                     obj  ) numval)
            ")"))
 (PRIN "}"))


;CREATION DE NOUVEAUX OBJETS:
; l'objet racine prédéfini, {obj:} (aucun champ), est lié à OBJ
; {NEW objet} retourne une copie de objet;
; {NEW objet 'var} retourne et lie à var une copie de objet;
; {NEW objet 'var '((nom val)…)} lie à var une extension de objet
; avec les nouveaux champs (nom val)… dont les méthodes d'accès
; sont crées automatiquement (dans le package <obj.package>:<var>):
; {nom var} retourne la valeur du champ nom de l'objet lié à var
; {nom var val} donne la valeur val au champ nom de l'objet lié à var
(DE :new args
 (LET ((model (:eval (CAR args))) ; argument 1 obligatoire
       (newmodel    (CADR args))  ; argument 2 optionnel
       (newchamps  (CADDR args))) ; argument 3 optionnel
  (COND ((NULL (CDR args))        ; {NEW model}
         (COPY model))
        ((NULL (CDDR args))       ; {NEW model 'newmodel}
```

```
      (SET newmodel (COPY model)))
     (T                          ; {NEW model 'newmodel '(newchamps)}
; création d'un nouveau modèle par extension de l'original:
(LETS ((newpackg (SYMBOL (:package model) newmodel))
      (numval  (VLENGTH (:objvals model)))
      (vals (MAKEVECTOR (+ numval (LENGTH newchamps)) ()))
      (noms (COPY vals)))
 ; copie des noms et des valeurs des champs du modèle:
 (BLTVECTOR noms 0 (:objvals (SYMEVAL (:package model))) 0 numval)
 (BLTVECTOR vals 0 (:objvals                    model  ) 0 numval)
 ; pour chacun des nouveaux champs:
 (MAPC (LAMBDA (champ)
       (LET ((method) (valeur))
        ; analyse syntaxique du champ:
        (COND ((ATOMP champ) ; champ = <method>
               (SETQ method champ
                     valeur nil)) ; valeur par défaut = nil
              ((AND (CONSP champ) ; champ = (<method> <valeur>)
                    (ATOMP (CAR champ)))  ; (<method>) = <method>
               (SETQ method (CAR champ)
                     valeur (EVAL (CADR champ))))
              (T ; mauvaise syntaxe: nettoyage du package
                 (MAPC 'REMOB (OBLIST newpackg))
                 (ERROR ':new "<champ>::=<m>|(<m><v>)" champ)))
        ; initialisation du nom et de la valeur du nouveau champ:
        (VSET vals numval valeur)
        (VSET noms numval method)
        ; création de la méthode d'accès au nouveau champ:
        (EVAL `(DE ,(SYMBOL newpackg method) (obj . arg)
                (:objval ,numval)))
        (INCR numval)))
      newchamps)
  ; variables contenant le nouveau modèle:
  (SET newpackg (TCONS newpackg noms))
  (SET newmodel (TCONS newpackg vals))) ))))

; fonction pour méthodes d'accès en lecture/écriture à un champ
; (obj et arg sont globales, voir 10° ligne ci-dessus)
(DE :objval (numval)
 (IF arg (VSET (:objvals obj) numval (CAR arg))
         (VREF (:objvals obj) numval)))


;LISTE ET CREATION DES METHODES:
; {METHOD obj} retourne la liste des noms des méthodes locales à obj
; {METHOD obj 'nom} affiche (pretty) la définition de la méthode nom
; {METHOD obj 'nom '(lvar body)} définit une nouvelle méthode de obj
; (le premier argument de la méthode doit être du type de obj)
```

```
(DE :method args
 (LET ((pkg (:package (:eval (CAR args)))) ; argument 1 obligatoire
       (nom                   (CADR args))   ; argument 2 optionnel
       (def                  (CADDR args)))  ; argument 3 optionnel
  (COND ((NULL (CDR args))      ; {METHOD obj}
         (REMQ () (MAPCAR (LAMBDA (atom)
                            (WHEN (TYPEFN atom) atom))
                          (OBLIST pkg))))
        ((NULL (CDDR args))      ; {METHOD obj 'nom}
         (EVAL `(PRETTY ,(GETFN pkg nom))))
        (T                       ; {METHOD obj 'nom 'def}
         (EVAL `(DE ,(SYMBOL pkg nom) ,@def))))))

; la méthode SET permet d'écrire plusieurs champs simultanément
; ex: {SET point '((x 12) (y 34))} --> {*:obj:point (x 12) (y 34)}
(DE :set (obj champs)
 (MAPC (LAMBDA (champ)
        (COND ((ATOMP champ) ; champ = <method>
               (SEND champ obj nil))
              ((AND (CONSP champ) ; champ = (<method> <valeur>)
                    (ATOMP (CAR champ)))  ; (<method>) = <method>
               (SEND (CAR champ) obj (EVAL (CADR champ))))
              (T (ERROR ':set "<champ>::=<m>|(<m><v>)" champ))))
       champs)
 obj)

; la méthode GET retourne une liste des valeurs des champs désignés
; ex: {GET point '(y x)} --> (34 12)
(DE :get (obj champs)
 (MAPCAR (LAMBDA (champ)
          (SEND champ obj))
         champs))
```

VERIFICATEUR DE REGLES DE DESSIN

par F.GUEZ et O.GIORDANO

```
[A:\]
/* ZONE root */
newzone(root)?


menu0 :-
   clear_scr,
   attribute(30),
   ((exit_box,clear_scr,fail) ; def_box(19,55,0,5)),
   pos(3,2),
   attribute(126),
   writes("CLASSEMENT DES ERREURS PAR TYPE"),
   exit_box,
   pos(15,7),
   writes("Erreurs de :"),
   pos(14,8),
   writes("--------------"),
   pos(25,10),
   writes("(L)argeur",31),
   menu1.

pass2 :-
   writes(" Troisieme passe ...",23),
   put(13),
   erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,3),
   r(X,Y,DX,DY,N,Q),
   r(X2,Y2,DX2,DY2,N,Q),
   recouvre(X1,X2,DX1,DX2,D3,D4),
   recouvre(Y1,Y2,DY1,DY2,D5,D6),
   not regle(N,N1,D3),
   not regle(N,N1,D4),
   not regle(N,N1,D5),
   not regle(N,N1,D6),
   retract(erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,3)),
   impri2(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,3).

pass6 :-
   writes("                    Erreurs supprimees :"),
   put(13),
   writes("                    ----------------------"),
   put(13),
   writes(" Deuxieme passe ...",23),
   put(13),
   erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N,4),
   r(X1,Y1,DX1,DY1,N,Q),
   r(X2,Y2,DX2,DY2,N,Q2),
   Q \= Q2,
   inter(X,X2,DX,DX2,D3),
   inter(Y,Y2,DY,DY2,D4),
   (not regle(N,N,1,D3) ; not regle(N,N,1,D4)),
   inter(X1,X2,DX1,DX2,D1),
   inter(Y1,Y2,DY1,DY2,D2),
   (not regle(N,N,1,D1) ; not regle(N,N,1,D2)),
   retract(erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N,4)),
   impri2(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N,4).
```

INTRODUCTION:

Le projet a pour but la conception en langage PROLOG d'un programme de Vérification de Règles de Dessin (VRD).

Ce vérificateur doit s'appliquer à des dessins de circuits intégrés réalisés en technologie CMOS. Un tel dessin sera soumis au programme sous forme d'une liste de rectangles, chaque rectangle étant décrit par les paramètres suivants:

- X : abscisse,
- Y : ordonnée,
- DX: étendue horizontale,
- DY: étendue verticale,
- N : niveau.

Nous étudierons dans ce rapport les performances du programme réalisé, ainsi que son fonctionnement.

I) Performances du programme:

I.1) Erreurs détectées:

Le programme détecte 4 types d'erreurs de dessin:

- erreurs de largeur,
- erreurs de distance,
- erreurs de recouvrement,
- erreurs d'intersection.

I.1.1) Erreur de largeur.

Une erreur de ce type est détectée si DX ou DY a une valeur insuffisante ("ou" inclusif).

I.1.2) Erreur de distance.

Cette erreur se produit lorsque deux rectangles se trouvent à une distance insuffisante l'un de l'autre.

I.1.3) Erreur de recouvrement.

Cette erreur peut avoir deux origines:

- soit un recouvrement interdit entre deux rectangles (par exemple contact/via).

- soit un débordement insuffisant d'un rectangle sur un autre dans le cas d'un recouvrement autorisé.

I.1.4) Erreur d'intersection.

Cette erreur se produit lorsque deux rectangles de même niveau sont superposés et que leur surface d'intersection est insuffisante.

```
pass3 :-
   r(X2,Y2,DX2,DY2,N,Q),
   r(X1,Y1,DX1,DY1,N1,Q1),
   Q \= Q1,
   (N = N1 ; (N = cont,N1 \= alu2) ; (N1 = cont,N \= alu2) ; (N = via,(N1 = alu1
    ; N1 = alu2)) ; ((N = alu1 ; N = alu2),N1 = via)),
   inter2(X1,X2,DX1,DX2),
   inter2(Y1,Y2,DY1,DY2),
   ((Q > Q1,Q2 is Q1) ; (Q < Q1,Q2 is Q)),
   retract(r(X2,Y2,DX2,DY2,N,Q)),
   retract(r(X1,Y1,DX1,DY1,N1,Q1)),
   asserta(r(X2,Y2,DX2,DY2,N,Q2)),
   asserta(r(X1,Y1,DX1,DY1,N1,Q2)),
   pass3.

pass4 :-
   writes(" Quatrieme passe ...",23),
   put(13),
   er1(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,2),
   N \= N1,
   r(X1,Y1,DX1,DY1,N1,Q1),
   r(X,Y,DX,DY,N,Q1),
   retract(er1(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,2)),
   impri2(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,2).

run :-
   attribute(30),
   (init ; en_tete1 ; fichier ; true),
   put(13),
   writes("              Pressez une touche pour continuer ..."),
   getc(C),
   menu0.

fichier :- vrai([r(0,0,1,3,alu1),r(0,3,4,4,alu1),r(0,7,2,5,alu1),r(1,4,2,2,cont)
   ,r(0,3,10,4,difn),r(4,2,2,6,poly),r(3,8,4,4,poly),r(7,0,2,10,alu1),r(3,9,4,3,
   alu1),r(4,9,2,2,cont),r(0,10,4,2,alu1),r(3,8,4,4,alu1),r(0,2,5,2,alu1),r(9,9,
   2,2,alu1)],0).

liste(R) :-
   er1(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,R),
   line_attr(30),
   tab(4),
   write(N),
   tab(4),
   writefi(X),
   writefi(Y),
   tab(8),
   write(N1),
   tab(4),
   writefi(X1),
   writefi(Y1),
   tab(8),
   err(R),
   put(13),
   fail.
```

I.2) Cas particuliers:

Le programme ne détecte pas d'erreur dans les cas particuliers suivants:

I.2.1) Distance.

Aucune erreur de distance n'est détectée entre deux rectangles de niveaux différents si ces derniers sont reliés électriquement.

I.2.2) Recouvrement.

Aucune erreur de recouvrement n'est détectée entre un rectangle R1 et un rectangle R2 si:

- R1 recouvre R2.
- il existe un rectangle R3 tel que:
  - .R3 est de même niveau que R1.
  - .R3 recouvre R2 sans erreur.

I.2.3) Intersection.

Aucune erreur de recouvrement n'est détectée entre deux rectangles de même niveau R1 et R2 s'il existe un rectangle R3 tel que:

- .R3 est de même niveau que R1 et R2.
- .Il n'existe pas d'erreur d'intersection entre R1 et R3.
- .Il n'existe pas d'erreur d'intersection entre R2 et R3.

I.3) Temps de calcul.

A titre indicatif, le traitement de l'essai "standard" de 14 rectangles présent dans le programme réclame environ   s pour détecter 7 erreurs.

I.4) Remarque sur les erreurs multiples.

Si pour un cas donné, le programme détecte une erreur de même nature en X et en Y, il ne signale qu'une erreur à l'utilisateur.

II) Fonctionnement du programme:

II.1) Représentation des objets manipulés par le programme;

II.1.1) Données de base.

Le circuit à tester est représenté par une clause définissant une liste, soit une structure de la forme:

fichier:-vrai([r(X1,Y1,DX1,DY1,N1),...,r(XP,YP,DXP,DYP,NP)]),

ou r(Xi,Yi,DXi,DYi,Ni) représente le i-ème rectangle de la liste.

Le traitement des cas particuliers (Cf. I.2) nécessite la traduction de cette liste de rectangles en une suite de clauses de la forme:

r(X1,Y1,DX1,DY1,N1,Q1)

.

.

r(XP,YP,DXP,DYP,NP,QP)

```
en_tete1 :-
   cursor(12,12),
   clear_scr,
   def_box(1,78,0,6),
   attribute(126),
   tab(16),
   writes("Verificateur de regles de dessin version 1.0."),
   put(13),
   tab(26),
   writes("Par F. Guez & O. Giordano"),
   put(13),
   tab(17),
   writes("Copyright (C) auteurs & D.E.A.  M.E.M.I. 1985",254),
   put(13),
   en_tete2,
   fail.

init :-
   retract(r(A,B,C,D,E,F)),
   fail.
init :-
   retract(erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,R)),
   fail.

en_tete2 :-
   line_attr(87),
   writes("Rectangle1"),
   tab(6),
   writes("X1:"),
   tab(3),
   writes("Y1:"),
   tab(4),
   writes("Rectangle2"),
   tab(5),
   writes("X2:"),
   tab(3),
   writes("Y2:"),
   tab(8),
   writes("Erreur"),
   exit_box,
   def_box(1,78,7,20),
   writes(" Premiere passe...",23),
   put(13).

regla(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1),3) :-
   recouvre(X1,X,DX1,DX,D1,D2),
   (regle(N,N1,3,D1) ; regle(N,N1,3,D2)),
   impri(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,3).
regla(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1),3) :-
   recouvre(X,X1,DX,DX1,D1,D2),
   (regle(N1,N,3,D1) ; regle(N1,N,3,D2)),
   impri(X1,Y1,DX1,DY1,N1,X,Y,DX,DY,N,3).
regla(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1),3) :-
   recouvre(Y,Y1,DY,DY1,D1,D2),
   (regle(N1,N,3,D1) ; regle(N1,N,3,D2)),
   impri(X1,Y1,DX1,DY1,N1,X,Y,DX,DY,N,3).
regla(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1),3) :-
   recouvre(Y1,Y,DY1,DY,D1,D2),
   (regle(N,N1,3,D1) ; regle(N,N1,3,D2)),
   impri(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,3).
regla(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N),4) :-
   inter(X,X1,DX,DX1,D5),
   inter(Y,Y1,DY,DY1,D6),
   regle(N,N,1,D5),
   regle(N,N,1,D6),
   impri(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N,4).

regla1(r(X,Y,DX,DY,N),r(X,Y,DX,DY,N)) :-
   not regle(N,N,1,DX),
   not regle(N,N,1,DY).
regla1(r(X,Y,DX,DY,N),r(X,Y,DX,DY,N)) :-
   regle(N,N,1,DY),
   impri(X,Y,DX,DY,N,X,Y,DX,DY,N,1).
regla1(r(X,Y,DX,DY,N),r(X,Y,DX,DY,N)) :-
   regle(N,N,1,DX),
   impri(X,Y,DX,DY,N,X,Y,DX,DY,N,1).
```

où les Qi sont des paramètres ajoutés lors du traitement (cf. II.2.3).

II.1.2) Erreurs.

Les erreurs sont représentées sous forme de clauses insérées en fin de programme.

Ces clauses sont de la forme:

erl(X1,Y1,DX1,DY1,N1,X2,Y2,DX2,DY2,N2,ERR).

où:

- (X1,Y1,DX1,DY1,N1) sont les paramètres du premier rectangle en cause.
- (X2,Y2,DX2,DY2,N2) sont les paramètres du deuxième rectangle en cause.
- Err est le code de l'erreur détectée, avec:

  .ERR = 1 => Erreur de largeur,
  .ERR = 2 => Erreur de distance,
  .ERR = 3 => Erreur de recouvrement,
  .ERR = 4 => Erreur d'intersection.

II.1.3) Règles.

Les règles sont exprimées sous forme de clauses du type:

règle(N1,N2,R,L) :- L<Min.

avec:

- N1 = niveau du premier rectangle,
- N2 = niveau du deuxième rectangle,
- R = code de la règle, où:
  .R=1 => règle de largeur,
  .R=2 => règle de distance,
  .R=3 => règle de recouvrement,
  .R=4 => règle d'intersection.
- L = paramètre à comparer à Min,
- Min= valeur minimale de la règle.

II.2) Recherche des erreurs:

II.2.1) Généralités.

Le programme de VRD procède en quatre passes consécutives:

- application stricte des règles (sans tenir compte des cas particuliers),
- examen des erreurs d'intersection,
- examen des erreurs de recouvrement,
- examen des erreurs de distance.

Pendant les trois dernières passes, le programme élimine les clauses d'erreurs relevant des cas particuliers décrits au paragraphe I.2).

```
regla2(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1)) :-
    inter2(Y,Y1,DY,DY1),
    disjoint(X,X1,DX,DX1,D1),
    regle(N,N1,2,D1),
    impri(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,2),
    fail.
regla2(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1)) :-
    disjoint(Y,Y1,DY,DY1,D2),
    disjoint(X,X1,DX,DX1,D1),
    regle(N,N1,2,D1),
    regle(N,N1,2,D2),
    impri(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,2),
    fail.
regla2(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1)) :-
    inter2(X,X1,DX,DX1),
    disjoint(Y,Y1,DY,DY1,D),
    regle(N,N1,2,D),
    impri(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,2),
    fail.
regla2(r(X,Y,DX,DY,N),r(X1,Y1,DX1,DY1,N1)) :-
    inter2(X,X1,DX,DX1),
    inter2(Y,Y1,DY,DY1).

regle(poly,poly,1,L) :- L < 2.
regle(difn,difn,1,L) :- L < 3.
regle(difp,difp,1,L) :- L < 3.
regle(cont,cont,1,L) :- L < 2.
regle(alu1,alu1,1,L) :- L < 2.
regle(alu2,alu2,1,L) :- L < 4.
regle(via,via,1,L) :- L < 2.
regle(poly,poly,2,L) :- L < 2.
regle(poly,difn,2,L) :- L < 1.
regle(poly,difp,2,L) :- L < 1.
regle(poly,cont,2,L) :- L < 2.
regle(difn,difn,2,L) :- L < 2.
regle(difn,difp,2,L) :- L < 2.
regle(difn,cais,2,L) :- L < 6.
regle(difn,poly,2,L) :- L < 1.
regle(difp,poly,2,L) :- L < 1.
regle(cont,poly,2,L) :- L < 2.
regle(difp,difn,2,L) :- L < 2.
regle(cais,difn,2,L) :- L < 6.
regle(cais,cais,2,L) :- L < 18.
regle(cont,cont,2,L) :- L < 2.
regle(alu1,alu1,2,L) :- L < 2.
regle(alu2,alu2,2,L) :- L < 2.
regle(via,via,2,L) :- L < 2.
regle(cont,via,2,L) :- L < 1.
regle(poly,difn,3,L) :- L < 2.
regle(poly,difp,3,L) :- L < 2.
regle(poly,cont,3,L) :- L < 1.
regle(difn,poly,3,L) :- L < 3.
regle(difn,cont,3,L) :- L < 1.
regle(difp,cont,3,L) :- L < 1.
regle(cais,difp,3,L) :- L < 2.
regle(alu1,cont,3,L) :- L < 1.
regle(alu1,via,3,L) :- L < 1.
regle(alu2,via,3,L) :- L < 1.
regle(difp,poly,3,L) :- L < 3.
regle(difp,difn,3,L).
regle(difn,difp,3,L).
regle(difn,cais,3,L).
regle(cais,difn,3,L).
regle(cont,via,3,L).
regle(via,cont,3,L).

disjoint(A,B,C,D,E) :-
    A > B + D,
    E is A - B - D.
disjoint(A,B,C,D,E) :-
    A + C < B,
    E is B - A - C.

recouvre(A,B,C,D,E,F) :-
    A >= B,
    A =< B + D,
    E is A - B,
    F is B + D - A - C.
```

Le programme indique à l'utilisateur le numéro de la passe en cours d'exécution, ainsi que les erreurs traitées à chaque étape (aussi bien en insertion qu'en retrait).

Ce logiciel étant expérimental, ces dispositions permettent à l'utilisateur de détecter des "bugs" éventuelles et offrent ainsi un meilleur suivi des opérations en cours

II.2.2) Détection des erreurs de:

II.2.2.1) Largeur.

Le VRD teste si les DX et DY de chaque rectangle de niveau N sont inférieur au Min de la règle:

règle(N,N,1,L).

avec L=DX ou L=DY. Si "règle" est vérifiée, une erreur de type 1 est générée.

II.2.2.2) Distance.

La clause "disjoint" calcule la distance entre deux rectangles de niveau N1 et N2 selon les x ou les y. Une erreur de type 2 est détectée si la règle:

règle(N1,N2,2,L).

est vérifiée pour la distance en x ET en y.

II.2.2.3) Recouvrement.

La clause "recouvre" calcule le débordement à droite et à gauche d'un rectangle de niveau N1 sur un rectangle de niveau N2. Si la règle:

règle(N1,N2,3,L).

est vérifiée pour le débordement à droite OU à gauche, une erreur de type 3 est générée. En outre, cette clause détecte les intersections interdites entre rectangles de niveaux différents.

II.2.2.4) Intersection.

La clause "inter" calcule les dimensions de l'intersection de deux rectangles de niveau N. Si la règle:

règle(N,N,4,L).

est vérifiée pour l'une des dimensions, une erreur de type 4 est générée.

II.2.3) Traitement des cas particuliers.

II.2.3.1) Deuxième passe.

Au cours de cette passe, le VRD parcours les erreurs d'intersection détectées durant la première passe. Il recherche pour chaque erreur un rectangle répondant aux conditions énoncées au paragraphe I.2.3).

II.2.3.1) Troisième passe.

Cette passe se déroule en deux phases:

- parcours des clauses "r(X,Y,DX,DY,N,Q)", comparaison des rectangles deux à deux, et changement du paramètre Q si intersection: dans ce cas Q=min(Q1,Q2).

```
inter(A,B,C,D,E) :-
   A =< B,
   A + C >= B,
   ((A + C < B + D,E is A + C - B) ; (A + C >= B + D,E is D)).
inter(A,B,C,D,E) :-
   A =< B + D,
   A > B,
   ((A + C > B + D,E is B + D - A) ; (A + C =< B + D,E is C)).

inter2(A,B,C,D) :-
   A =< B,
   A + C >= B.
inter2(A,B,C,D) :-
   A =< B + D,
   A > B.

test(T,T1) :-
   regla2(T,T1),
   regla(T,T1,P),
   fail.
test(T,T1).

impri(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,ERR) :-
   not erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,ERR),
   line_attr(30),
   tab(4),
   write(N),
   tab(4),
   writefi(X),
   writefi(Y),
   tab(8),
   write(N1),
   tab(4),
   writefi(X1),
   writefi(Y1),
   tab(8),
   err(ERR),
   asserta(erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,ERR)),
   put(13).

impri2(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,ERR) :-
   line_attr(31),
   tab(4),
   write(N),
   tab(4),
   writefi(X),
   writefi(Y),
   tab(8),
   write(N1),
   tab(4),
   writefi(X1),
   writefi(Y1),
   tab(8),
   err(ERR).

verif(T,[T1|R]) :-
   test(T,T1),
   verif(T,R).
verif(T,[]).

err(2) :- write(distance).
err(3) :- write(recouvrement).
err(1) :- write(largeur).
err(4) :- write(intersection).

vrai([],Q) :-
   put(13),
   (pass6 ; pass3 ; pass2 ; pass4),
   put(13),
   fail.
vrai([T|S],Q) :-
   asser(T,Q),
   regla1(T,T),
   verif(T,S),
   O is Q + 1,
   vrai(S,O).

asser(r(X,Y,DX,DY,N),Q) :- asserta(r(X,Y,DX,DY,N,Q)).
```

Cette phase peut servir d'extracteur d'équipotentielles, mais ne prend pas en compte les transistors (le drain, la source et la grille sont considérés comme appartenant à la même équipotentielle.

- parcours des erreurs de distance; si les rectangles en cause sont de niveaux différents et ont le même paramètre Q, l'erreur est supprimée.

II.2.3.3) Quatrième passe.

Au cours de cette passe, le programme parcours les erreurs de recouvrement. Il recherche pour chaque erreur un troisième rectangle vérifiant les conditions du I.2.2).

II.2.3.4) Remarque.

Au cours de la première passe, chaque élément de la liste des rectangles est mis sous forme de clause, le paramètre Q étant initialisé à l'ordre d'insertion du rectangle (Q=0 pour le premier, Q=1 pour le deuxième, etc...).

III) Mode d'emploi:

III.1) Données:

La liste des rectangles doit être présente dans le programme sous la forme décrite au paragraphe II.1).

III.2) Départ:

taper: run?   (sous Xlog)

III.3) Résultats:

Un premier affichage va permettre le suivi des opérations de vérification.

En fin d'opération, un menu va permettre de lire toutes les erreurs détectées.

Cinq choix sont offerts:

- (L) consulte les erreurs de largeur,
- (D) consulte les erreurs de distance,
- (R) consulte les erreurs de recouvrement,
- (I) consulte les erreurs d'intersection,
- (Q) retour à Xlog.

Chaque erreur est représentée par l'affichage des niveaux et des coordonnées des deux rectangles en cause.

```
menu1 :-
    pos(25,12),
    writes("(R)ecouvrement.",29),
    pos(25,14),
    writes("(D)istance.",28),
    pos(25,16),
    writes("(I)ntersection.",27),
    menu2.

menu2 :-
    pos(45,19),
    writes("(Q)uitter ."),
    get(Choix),
    clear_scr,
    (affiche(Choix) ; menu0).

affiche(X) :-
    (X = "l" ; X = "L"),
    def_box(10,68,0,5),
    attribute(127),
    pos(20,2),
    writes("ERREURS DE LARGEUR"),
    exit_box,
    efface,
    imprime(1).
affiche(X) :-
    (X = "d" ; X = "D"),
    def_box(10,68,0,5),
    attribute(124),
    pos(20,2),
    writes("ERREURS DE DISTANCE"),
    exit_box,
    efface,
    imprime(2).
affiche(X) :-
    (X = "r" ; X = "R"),
    def_box(10,68,0,5),
    attribute(125),
    pos(16,2),
    writes("ERREURS DE RECOUVREMENT"),
    exit_box,
    efface,
    imprime(3).
affiche(X) :-
    (X = "i" ; X = "I"),
    def_box(10,68,0,5),
    attribute(123),
    pos(16,2),
    writes("ERREURS D'INTERSECTION"),
    exit_box,
    efface,
    imprime(4).
affiche(X) :-
    (X = "q" ; X = "Q"),
    cursor(5,5),
    stop.

efface :-
    (exit_box ; true),
    def_box(10,68,6,20),
    attribute(126),
    clear_scr,
    cadre0,
    exit_box.

imprime(ERR) :-
    erl(X,Y,DX,DY,N,X1,Y1,DX1,DY1,N1,ERR),
    def_box(10,68,6,20),
    pos(15,4),
    writefi(X),
    pos(15,6),
    writefi(Y),
    pos(15,8),
    writefi(DX),
    pos(15,10),
    writefi(DY),
    pos(44,4),
    writefi(X1),
    pos(44,6),
    writefi(Y1),
```

III.4) Remarque:

Après un "Quitte" ou un arrêt du programme, la lecture des erreurs détectées à cet instant se fait par:

menu0? (sous Xlog)

IV) Performances:

L'amélioration des temps de traitement n'a pas été pour nous l'objectif principal de notre travail. En effet, la complexité du problème et le grand nombre de cas particuliers possibles entrainent une multitude de tests qui vont ralentir l'exécution.

Il nous est apparu plus important de réaliser un logiciel couvrant le plus de cas possibles et laissant à l'utilisateur une trace de la "réflexion" du programme. En effet, la possibilité de vérifier et trier les résultats évite la prise en compte d'erreurs fictives.

IV.1) Structure des données:

Le choix de l'introduction des rectangles sous forme de liste a été dicté par un souci de compacité, mais aussi pour améliorer le temps de calcul dans la première passe où les rectangles sont examinés deux par deux. Une procédure réccurente d'extraction et de traitement des têtes de liste est la solution la plus rapide.

IV.2) Exécution:

Il est préférable de tester d'abord si deux rectangles sont disjoints et de vérifier si les règles de distance sont respectées. On évite ainsi des traitements inutiles des règles de superposition.

Deux clauses permettent de déterminer s'il y a intersection, car l'une d'elle fournit les dimensions de la surface de superposition, tandis que l'autre ne donne aucune information d'encombrement. Un choix judicieux entre ces deux clauses permet d'éviter les passages de paramètres inutiles et donc de gagner du temps.

L'initialisation des paramètres Q pourrait faire l'objet d'un traitement plus rapide au cours de la troisième passe. On éviterait une multiplication des tests d'intersection par un "pré-marquage" des lignes équipotentielles dans la première passe lors du test de distance.

IV.3) Nombre de tests.

Une optimisation possible du VRD consisterait à optimiser le nombre de comparaisons de rectangles effectuées au cours des différents tests et, à cet effet, plusieurs solutions ont été proposées.

En particulier, une de ces solutions proposait de découper la surface du circuit en sous-blocs testés individuellement. Il parait préférable de ne soumettre au VRD que des fichiers de petite taille, respectant par exemple, le découpage en blocs fonctionnels du circuit global.

Si vous utilisez TURBO-PASCAL, vous savez certainement que les programmes utilisent, pour tourner, une bibliothèque qui a une longueur de 8 à 10 Ko et qui se place automatiquement en début de programme lorsque vous les compilez en extension .COM.

Mais quand vous devez sauvegarder des mini-applications nombreuses sur une disquette unique, les bibliothèques prennent à chaque application, une place en mémoire non négligeable.

Pour éviter ces redondances, il suffit de compiler le petit programme suivant avec la bibliothèque sous le nom bien connu de RUN.COM et de l'intégrer à la disquette d'applications.

Les programmes PASCAL seront compilés avec l'extension .CHN en choisissant l'option H afin de ne pas intégrer la bibliothèque. Ainsi, ils pourront être chaînés au programme RUN et tourner normalement.

```
Program Run;

Var Fichier:String[14];
    Logiciel:File;


Begin
   If ParamCount = 0 Then
   Begin
      Write ('Entrez le nom du fichier (Extension .CHN Par defaut) : ');
      ReadLn (Fichier);
   End
   Else Fichier := ParamStr(1);
   If Pos ('.',Fichier) = 0 Then Fichier := Fichier + '.CHN';
   Assign (Logiciel,Fichier);
   {$I-}
   Chain (Logiciel);
   {$I+}
   If IOResult <> 0 Then WriteLn ('ERREUR: FICHIER NON TROUVE ',^G);
End.
```

CONCLUSION:

Le programme proposé détecte toutes les erreurs de dessin. En revanche, il est possible que certaines configurations géométriques autorisées aboutissent à la génération d'une erreur.

La structure du programme révèle d'ailleurs la démarche suivie lors de sa conception: dans une première version, détecter toutes les erreurs, puis, dans les étapes suivantes, traiter ces erreurs de manière plus fine à chaque étape.

L'étape suivante, dans l'évolution du VRD, serait donc de reprendre le programme avec une vue d'ensemble, et d'en réviser la structure.

```
   pos(44,8),
   writefi(DX1),
   pos(44,10),
   writefi(DY1),
   imprimeO(N,N1).

imprimeO(N,N1) :-
   pos(19,2),
   write(N,5),
   pos(49,2),
   write(N1,5),
   pos(30,13),
   getc(0),
   exit_box,
   fail.

cadreO :-
   pos(5,2),
   writes("RECTANGLE 1 :"),
   pos(10,4),
   writes("X ="),
   pos(10,6),
   writes("Y ="),
   pos(10,8),
   writes("DX="),
   pos(10,10),
   writes("DY="),
   cadre1.

cadre1 :-
   pos(35,2),
   writes("RECTANGLE 2 :"),
   pos(40,4),
   writes("X ="),
   pos(40,6),
   writes("Y ="),
   pos(40,8),
   writes("DX="),
   pos(40,10),
   writes("DY="),
   pos(17,12),
   writes(">>> PRESSEZ UNE TOUCHE <<<",127).

r(4,2,2,6,poly,0).
r(3,8,4,4,poly,0).
r(4,9,2,2,cont,0).
r(3,9,4,3,alu1,0).
r(7,0,2,10,alu1,0).
r(9,9,2,2,alu1,0).
r(3,8,4,4,alu1,0).
r(0,10,4,2,alu1,0).
r(0,7,2,5,alu1,0).
r(0,3,4,4,alu1,0).
r(0,3,10,4,difn,0).
r(1,4,2,2,cont,0).
r(0,0,1,3,alu1,0).
r(0,2,5,2,alu1,0).

erl(7,0,2,10,alu1,9,9,2,2,alu1,4).
erl(4,2,2,6,poly,0,3,10,4,difn,3).
erl(0,7,2,5,alu1,3,8,4,4,alu1,2).
erl(0,7,2,5,alu1,3,9,4,3,alu1,2).
erl(0,3,4,4,alu1,3,8,4,4,alu1,2).
erl(0,0,1,3,alu1,0,2,5,2,alu1,4).
erl(0,0,1,3,alu1,0,3,4,4,alu1,4).
erl(0,0,1,3,alu1,0,0,1,3,alu1,1).
```

Tout individu sachant lire, écrire et compter peut tirer profit d'un interprète APL mis à sa disposition sur un micro-ordinateur ou à partir d'un terminal. Il n'est pas nécessaire d'être informaticien.

L'APL peut être utilisé à différents niveaux de complexité, pour faire des choses extrèmement simples ou des choses très élaborées.

Au niveau le plus élémentaire, l'APL se comporte comme une calculette d'une puissance et d'une facilité d'emploi incomparables.

Facilité d'emploi: il suffit de taper

```
2 + 2
```

pour obtenir la réponse: 4 . Si on frappe:

```
PI ← 3.14
```

on stocke la valeur 3.14 dans un registre appelé "PI". Si on frappe alors:

```
2 x PI
```

la réponse sera: 6.28

Puissance: les 4 opérations, la racine carrée, l'exponentielle, le logarithme, les fonctions trigonométriques, etc... sont exécutées instantanément avec une précision supérieure à 15 décimales. Le nombre de registres est quasi illimité (création de variables). Il esr recommandé au débutant d'user et d'abuser du mode "calculette". S'il a pris l'habitude de placer sous son coude gauche un tableau récapitulatif des symboles opératoires, il pourra effectuer facilement toutes les opérations élémentaires qui lui seront nécessaires. Il mémorisera ainsi rapidement et sans même s'en rendre compte les principaux symboles utilisés, et bientôt, il pourra ranger le tableau dans son tiroir pour ne l'en ressortir qu'en cas de besoin.

EXEMPLE DE CALCUL

A titre d'exemple, voyons comment calculer les racines d'une équation du second degré:

```
a.x2 + b.x + c = 0
```

pour les valeurs a=1 b=1 c=-2

Nous allons commencer par mémoriser les coefficients en créant trois variables A B et C. Nous frappons:

```
A ← 1
B ← 1
C ← -2
```

Notons au passage que l'APL fait la différence entre le moins (-) symbole de l'opération de soustraction et le moins (-) préfixe des nombres négatifs. Après quoi nous allons calculer le déterminant:

```
(B * 2) ÷ (4 x A x C)
```

La réponse de l'APL est: 9. Il est positif, nous aurons donc deux racines réelles. Nous allons ranger la racine carrée de ce déterminant dans une variable que nous appelerons D:

```
D ← ((B * 2) ÷ (4 x A x C)) * 0.5
```

Vérifions la valeur de D en tapant tout simplement: D. La réponse est 3. Nous n'en sommes pas étonnés, et nous continuons en calculant la valeur de la première racine:

```
((-B) + D ) ÷ (2 x A)
```

L'APL nous renvoie la valeur: 1 et la deuxième racine:

```
((-B) - D ) ÷ (2 x A)
```

L'APL nous renvoie la valeur: -2.

Si le déterminant avait été négatif, nous aurions rangé dans D la racine carrée de son opposé:

```
D ← ((4 x A x C) ÷ (B * 2)) * 0.5
```

et nous aurions calculé successivement la partie réelle:

```
(-B) ÷ (2 x A)
```

et la partie imaginaire:

```
D ÷ (2 x A)
```

des deux racines conjuguées.

ET APRES ...

A ce point, une remarque vient naturellement à l'esprit: si nous voulons les racines de l'équation pour d'autres valeurs de A, B et C, nous allons recommencer à taper la même série de commandes (laquelle, notons-le, comporte une variante selon le signe du déterminant). N'est-il pas possible d'enregistrer une fois pour toutes cette série de commandes et de la faire exécuter séquenciellement sans avoir à intervenir autrement qu'en entrant les valeurs des données?

La réponse est: oui. Une suite de commandes (ou instrucions) pré-enregistrées s'appelle une "fonction" APL. Mais là, nous franchirons un pas dans l'escalade de la complexité en passant du mode "calculette" à la programmation proprement dite.

Pour l'instant, nous sommes encore loin d'avoir épuisé toutes les possibilités du mode "calculette" et nous verrons prochainement comment manipuler des variables multidimensionnelles, vecteurs, tableaux... ■



21 Forth7 24.07 20H34
I'M COMING BACK !
Alors plus moyen de trouver des trucs en FORTH dans ce forum lethargique ?
Faut encore que je me devoue...

Un mot (une primitive i.e) est devenu inutile dans le dictionnaire ?

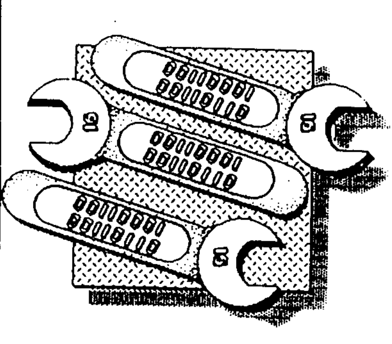
Faites FORBID mot
- 'mot' est detruit dans le dico
- les recherches sont plus rapides
- il n'encombre plus les listes
- toutes les anciennes definitions ayant utilise 'mot' restent valides

Definition en FIG-FORTH -->

```
A LA TRAPPE...
( FORBID suppression d'un mot )
FORTH DEFINITIONS DECIMAL
: FORBID [COMPILE] ' NFA CONTEXT @
        BEGIN
          2DUP @ = 0= WHILE
          @ PFA LFA
        REPEAT
        SWAP PFA LFA @ SWAP ! ;
;S
```

comparez FORBID et FORGET ! ■

SOUS LE CAPOT
DU 83-STANDARD: l'éditeur



BASES DE DONNEES
ET SYSTEMES EXPERTS: suite

THEORIE DE L'INFORMATION
THEOREME DE SHANNON

Par convention, la quantité d'information nécessaire pour transmettre un symbole binaire, dont, à chaque instant, la pro- babilité d'apparition est de 1/2, est égale à un bit.

La quantité d'information nécessaire pour transmettre un message de l symboles, Si, d'un alphabet prédéfini (S1 S2 ... Si ... Sn), de probabilité d'apparition constante, notée pi, est :

$$Q_O = \sum_{i=1}^{l} (-p_i * l(p_i)) \quad l(p) : \text{logarithme à base } 2$$

HYPOTHESES DE HUFMAN

1 Les symboles sont appelés caractères.
2 Leur probabilité d'apparition ne dépend que du langage utilisé.
3 Le codage se fait caractère par caractère.
4 Le décodage aussi.

Il convient donc de noter que :
1 L'algorithme est entièrement prédifini avant la réception du premier caractère : il n'est pas nécessaire d'avoir reçu, partiellement ou non, le message à décoder ; la mémoire nécessaire est donc très faible.
2 La deuxième hypothèse n'est que partiellement vérifiée : le texte déjà décodé permet parfois de bien prévoir la suite. L'algorithme ignore ces redondances.
3 Le langage de référence n'est pas spécifié : il peut s'agir
- d'une langue naturelle : français, anglais, allemand, ...
- d'une langue artificielle, de programmation ou non;
- d'un message formaté : météo, séquences morse, séquences de mesures ;
- d'une entité logique graphique ou sonore déjà décomposée en symboles ou nombres : tableau de valeurs, image ou son, synthétiques ou non, etc.....

ALGORITHME DE HUFMAN

Il faut disposer de l'histogramme normalisé des probabilités, c'est-à-dire la fréquence d'apparition de chaque symbole, leur somme étant égale à 1. Puis :

1 Les ranger par probabilités décroissantes.

2 Construire un arbre binaire de décodage, dont, à chaque noeud, les fréquences d'apparition de 0 et de 1 soient aussi voisines que possible.

3 Chaque caractère est remplacé par la suite binaire de son arbre de décodage, de longueur variable.

Le codage engendre donc un flot continu de bits et le décodage se fait bit à bit, d'après l'arbre de décodage.

Plusieurs symboles ont la même séquence finale : le décodage est donc autosynchronisant ; on peut commencer à décoder au milieu du message. Au bout d'un certain temps le décodage sera exact et le restera.

La mise en oeuvre nécessite la construction d'un arbre binaire, le codage se faisant en asociant un bit arbitraire à chaque branche. Il peut être :
- naturel : 1 à droite (fréquences faibles)
0 à gauche.
- équilibré : certains noeuds sont permutés pour équilibrer 0 et 1.
- crypté : l'ordre des noeuds est fixé, puis chacun d'eux est affecté d'une variable binaire indiquant s'il faut l'inverser.

EXEMPLE

L'exemple choisi est celui de la langue française. Les statistiques résultent d'un comptage sur 60.400 caractères dont 9.880 blancs et 1.440 lettres accentuées, et d'un autre, sur 100.000 caractères, cité dans le livre de Mr **Roubaty** "ABC du décryptage", paru aux éditions Masson.

1er comptage

î	1	...	30	4	44	è	120
ï	1	;	30	5	44	-	140
ç	2	:	30	6	44	L	170
%	2	(	40	7	44	à	170
+	2	)	40	8	44	.	340
/	2	0	44	9	44	M	570
!	2	1	44		70	'	590
	10	2	44		70	,	620
â	20	3	44	e	90	é	1040

2ème comptage

ƀ	9880	g	959	l	5867
------		f	1030	u	6144
k	34	q	1201	r	6457
z	148	v	1585	t	7195
y	262	p	2891	n	7381
x	390	m	2921	i	7437
j	431	c	3323	s	8119
h	859	d	3656	a	8192
b	881	o	5390	e	17230

M : Majuscule non précédée de .
L : Fin de paragraphe.

L'arbre de décodage se compose de noeuds et des symboles terminaux que sont les caractères.

Appelons *poids* le nombre de fois qu'un noeud est atteint, et *volume* le nombre total de branches qui restent à parcourir après avoir atteint un noeud. Si un noeud *n* résulte de la fusion de deux noeuds *n1* et *n2* , on a :

$$p = p1 + p2 \qquad ; \quad v = v1 + v2 + p ;$$

Un algorithme de construction de l'arbre de codage est le suivant :
1 Ranger les symboles par probabilité croissante.
2 Empiler les deux premiers symboles.
3 Calculer poids et volume du prochain noeud, et les empiler.
4 Le dernier noeud est l'origine de l'arbre.
Après alignement du premier comptage sur le deuxième (coefficient 2,1110407), la partie supérieure de la pile de Hufman ressemble à ceci :

La longueur moyenne d'un caractère est de :
556463 / 127388 = 4,3682 bits / caractère
soit un résultat légèrement supérieur à celui de la formule de **Shannon**, en raison de l'apairage imparfait

des poids. Si on renonce à la propriété autosynchronisante du code, il est possible de grouper les bits par 4, ce qui donne :

4 bits : ƀeainstrulodcm A B
8 bits : A év,'qMfghb.jxàL-
8 bits : B yéê<<>>z012345678 C
12 bits : C 9():;k âô?!/+%ç D
16 bits : D

60	59	61	127388	556463		o	42	47	9971	17038
58	37	60	73112	267829				s	8110	0
56	55	59	54276	161246				a	7833	0
54	ƀ	58	41002	116359		41	d	46	7624	16334
53	52	57	31012	78358				i	7436	0
e	51	56	29167	43743				n	7381	0
50	49	55	25109	63227				t	7195	0
48	47	54	21243	74259		c	40	45	6641	13009
		ƀ	20857	0				r	6457	0
s	a	53	15951	15951				u	6144	0
46	i	52	15060	31394				l	5867	0
		e	14591	0		p	m	44	5812	6812
n	t	51	14576	14576		39	38	43	5440	18914
45	r	50	13098	25107				o	5399	0
u	l	49	12011	12011		37	é	42	4601	7047
44	43	48	11252	35978		36	35	41	3968	8710

La longueur moyenne d'un caractère est alors de 5,0518 b/c, soit 16% de plus.

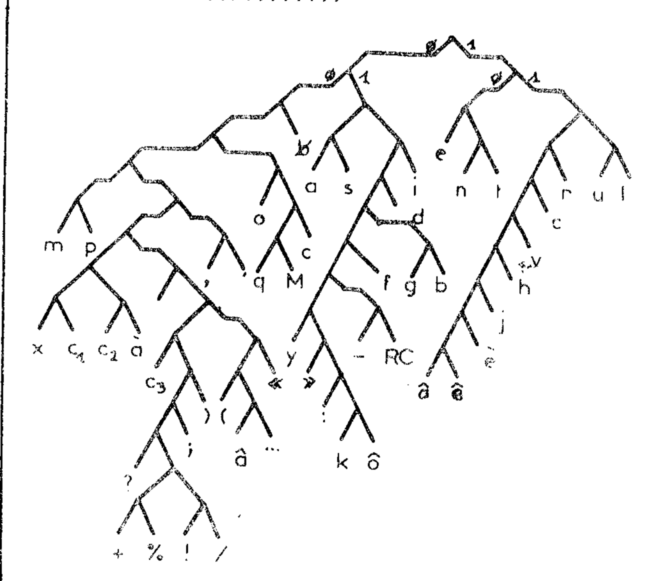
REPRESENTATION DE L'ARBRE DE DECODAGE

Celle-ci peut être pré, post ou infixée, c'est-à-dire parenthésée : le début de l'arbre serait alors représenté ainsi :

(((((îî)ç)(%+)) (/!))(?ô))â

En notation postfixée, la représentation complète de l'arbre est la suivante:

îî,ç%+!,,,,?ô,kâ...;,:,(),0123456789,,,,,,,,êè,y-L,,à,,xj,,,,,,,h,bgf,qM' , ,,v,,,é,,,pm,cd,,o,,lur,tni,as,,,,,e,,ƀ,,,,,,,,



La virgule représente l'apairage de deux noeuds et le symbole , est entre deux blancs.

APPLICATION

1) Traitement de texte et programmation

Tel quel, l'algorithme n'est applicable qu'au codage et décodage bit à bit de texte de statistique connue, mais les adaptations sont nombreuses. Il n'est pas interdit de conserver sa jugeotte : plutôt supprimer que de coder les blancs en fin de ligne ! Si les répétitions sont nombreuses, il est vivement conseillé de créer un symbole "répétition", ou un symbole banalisé représentant un mot ou une expression.

Pour coder un texte source, il n'est pas mauvais de faire, à la main ou par programme, un comptage de mots-clés : les plus fréquents pourront ainsi être codés sur un octet !

2) Fonction continue

a) Echantillonnage régulier.

Il faut définir une formule d'extrapolation, qui peut être simplement : $S_{n+1} = S_n$, puis ne coder que l'écart entre la mesure et l'extrapolation.

b) Echantillonnage non régulier.

En temps différé, il est possible, une fois définis un écart maximum et une formule d'interpolation, de ne transmettre que les points anguleux. Ne sont conservés que les points anguleux, c'est-à-dire les couples (l,v) : longueur et valeur. Les points intermédiaires sont restitués par interpolation. En temps réel, il faut un tampon de sortie pour faire ceci.

Ce n'est pas dévoiler un secret que de mentionner qu'aucune de ces techniques n'a été utilisée pour le Compact-Disk : le son a été codé, en absolu, linéairement sur 16 bits, c'est-à-dire que le rapport signal à bruit de quantification varie beaucoup avec le niveau : 12 bits auraient suffi avec un codage pseudo- exponentiel.

Aucun algorithme d'extrapolation ou de suppression d'échantillon n'a été employé. Par contre, la suppression des erreurs est irréprochable, au prix d'une redondance de deux.

c) Télévision

Il y a ici une énorme redondance, que ce soit entre points, lignes et images. De plus, les plans sont souvent fixes ou quasiment : défilement, effet zoom.

Comme le plus souvent un mobile se déplace sur fond fixe, il suffit de transmettre ou de chercher en mémoire le fond démasqué. Rien n'interdit de penser qu'avec de nouveaux processeurs il serait possible d'obtenir une télévision numérique haute définition moins gourmande en information que les appareils actuellement à l'étude. Moralité : les laboratoires ont encore du pain sur la planche !

Une petite anecdote pour conclure : lors de son passage entre Saturne et Uranus, la sonde Voyager II a reçu un programme de compression de l'information qui a ainsi fait gagner un facteur 2 dans le débit de transmission.

■

Le noyau F83 fourni par JEDI n'assigne pas les touches fonctions et curseur du clavier. Ceci va nous permettre de le reconfigurer à notre guise: à une touche sera affecté un mot FORTH, et cette affectation pourra être modifiée. Nous disposerons ainsi d'un clavier standard, d'un clavier éditeur, etc...

La ré-affectation peut se faire dans un programme. Par exemple, les touches F1 à F10 auront des fonctions différentes dans le MENU, l'AFFICHAGE ou la RECHERCHE.

Il existe cependant des limitations:

- une touche pré-définie sous MSDOS (ou un programme tel que NEWKEY) ne peut plus être redéfinie;

- la taille de la table des codes dans MSDOS est limitée.

La modification des touches se fait facilement à l'aide des codes d'échappement ANSI dont le fichier ANSI.SYS est fourni avec MSDOS (rappelons qu'il doit figurer dans CONFIG.SYS). La syntaxe en est:

ESC [<ancien-code>;<couveau-code>p

ou bien

ESC [<ancien-code>;"chaîne";...;<...>p

Les codes des touches fonctions sont en ASCII étendu, "ancien-code" est toujours de la forme 0;NN ; voir écrans 16 et 17.

Le programme proposé commence par s'occuper de l'affichage. Il suffit d'envoyer à l'écran les codes vidéo convenables. Les mots curseur seront utiles pour l'affectation des touches de déplacement.

Toutes les touches n'ont pas été affectées ici: elles sont laissées libres pour une utilisation personnelle.

La méthode s'applique bien entendu à toutes les touches du clavier. Mais si vous midifiez une touche alphanumérique, vous risquez de grosses surprises.

Les écrans ombre contiennent tous les renseignements nécessaires à la syntaxe et l'utilisation des mots créés.

par A.JACCOMARD 29oct86

```
    1
\ VIDEO et CLAVIER.

  2  LOAD  \ vidéo
  5  LOAD  \ clavier.
  7  LOAD  \ codes clavier "standard".
  8  LOAD  \ codes clavier "éditeur".
  9  LOAD  \ configuration du système.

 DEFER CLAVIER

  ' CLAV-STD IS CLAVIER    CLAVIER  \ configuration standard.

  5 VIEWS CLAVIER.BLK \ place ce fichier dans VIEW-FILES
```

```
    10
\ VIDEO et CLAVIER.                                    JaD08Oct86

   Ce pgm permet le contrôle de l'affichage ( vidéo inversée,
caractères gras, ... ) et des commandes du curseur.
   Les séquences d'échappement ANSI sont utilisées: le fichier
ANSI.SYS doit donc être sur le disque, et la commande
   DEVICE = ANSI.SYS  placée dans le fichier CONFIG.SYS.
   La syntaxe en est:
pour une modification de touche
   ESC [ ancien_code ; nouveau_code p
pour une affectation de touche
   ESC [ code;"chaîne"; ... p

Ex.: ESC[65;81p  remplace "A" par "Q"  (c'est assez dangereux)
       ESC[0;79;"CAPACITY 1- LIST";13p   l'appui sur la touche
   CTL-Fin fait afficher le dernier écran du fichier courant.
```

```
    2
\ VIDEO.                                               15Oct86JaD

: VDO   (S n --- : code, cf. "ombre )
   27 EMIT 91 EMIT   EMIT   109 EMIT ; \ ESC["n"m

: ATTB   (S n --- : attribut d'écran, cf. "ombre" )
   27 EMIT 91 EMIT  EMIT  104 EMIT ; \ ESC["n"h

: RET80C ( retour au mode 80X25, couleurs )
   51 ATTB ;

: RET80N ( retour au mode 80X25, monochrome )
   50 ATTB ;

-->
```

```
    11
\ VIDEO.                                               15Oct86JaD

VDO    (S n --- )  n = 48 : affichage normal ( par défaut );
                   n = 49 : caractères gras ;
                   n = 52 :      "     soulignés ;
                   n = 53 :      "     clignotants ;
                   n = 55 : vidéo inversée ;
                   n = 56 : caract. non affichés.

ATTB   (S n --- )  n = 0 : écran 40X25, monochrome ;
                   n = 1 :    "     "    couleurs ;
                   n = 2 :    "  80X25, monochrome ;
                   n = 3 :    "     "    couleurs ;
                   n = 4 : 320X200, couleurs ;
                   n = 5 :    "      monochrome ;
                   n = 6 : 640X200, monochrome.
```

```
3
\ CURSEUR.                                                  15Oct86JaD

: CURS (S n --- )  27 (CONSOLE) 91 (CONSOLE) (CONSOLE) ;

: CLS  50 CURS 74 (CONSOLE) ;  \ ESC[ 2J

: C-HT (S --- )
  49 CURS  65 (CONSOLE) ;  \ ESC[ "1" A
: C-BAS (S --- )
  49 CURS  66 (CONSOLE) ;  \ ESC[ "1" B
: C-DR (S --- )
  49 CURS  67 (CONSOLE) ;  \ ESC[ "1" C
: C-GCH (S --- )
  49 CURS  68 (CONSOLE) ;  \ ESC[ "1" D

-->
```

```
12
\ CURSEUR.                                                  15Oct86JaD

CLS  comme DARK, sans modifier les attributs d'écran.

CURS  n = 115 : sauvegarde position curseur ;
      n = 117 : retour à position sauvegardée.
C-HT  déplace curs. vers le haut de une ligne ;

C-BAS  de même, vers le bas ;

C-DR   de même, vers la droite, de un caract. ;

C-GCH  de même, vers la gauche.
```

```
4
\ CURSEUR.                                                  15Oct86JaD

: EFF-DEB   49 CURS  74 (CONSOLE) ;   \ ESC [ 1J

: EFF-FIN   74 CURS ;              \ ESC [ J

: EFF-LIG   75 CURS ;              \ ESC [ K

: EFF-D-LIG  49 CURS  75 (CONSOLE) ;   \ ESC [ 1K

: EFF-F-LIG  50 CURS  75 (CONSOLE) ;   \ ESC [ 2K
```

```
13
\ CURSEUR.                                                  15Oct86JaD

EFF-DEB   efface l'écran depuis le coin supérieur g. jusqu'au
          curseur inclus.
EFF-FIN   efface l'écran depuis curseur inclus jusqu'à fin écran

EFF-LIG   efface la ligne courante.

EFF-D-LIG efface début de ligne à curseur inclus.

EFF-F-LIG efface de curseur à fin de ligne.
```

```
5
\ Reconfiguration clavier : ,& et MODIF.                    JaD08Oct86

: ,&  (S --- ) \ place le texte suivant ds dict.; "&" délim.
  ASCII &  PARSE   32 SKIP  \ supprime les "blancs" en tête.
      HERE  PLACE ;     \ place texte en HERE.

: MODIF (S --- ) \ modif. une touche.
  ,&  HERE COUNT BOUNDS  27 EMIT    91 EMIT  \ pr ESC[
      ?DO  I C@ EMIT  LOOP ;

-->
```

```
14
\ Reconfiguration clavier : ,& et MODIF.                    15Oct86JaD

,&     est un alias du mot ," du noyau: ce dernier est inutili-
       sable ici, à cause du délimiteur " . Prend le texte
       suivant du flot d'entrée (clavier ou bloc) et le place
       à HERE.

MODIF  permet une modification rapide d'une touche, si on
       connait son code. Par exemple, pour affecter à la touche
       CTL-HOME, de code étendu 0;119, la fonction LISTING,
        entrez
          MODIF 0;119;"LISTING";13p suivi de ENTER,
       et la touche CTL-HOME est ré-affectée.

   UNE TOUCHE AFFECTEE SOUS MS-DOS NE PEUT PLUS ETRE RE-AFFECTEE.
```

```
6
\ Reconfiguration clavier : CONFIG.                         15Oct86JaD

: CONFIG  (S n -- : nbr de touches à modifier )
  1 ?ENOUGH  CREATE  DUP C,  0
    ?DO
    ,&  HERE  C@ 1+ ALLOT  \ mise à jour DP.
    LOOP
  DOES>
  COUNT  0              \ le mot créé dépose son adr sur la pile.
   ?DO
   27 EMIT  91 EMIT            \ pour ESC[
    COUNT  2DUP + -ROT  BOUNDS  \ calcule adr car. suivant.
    ?DO  I C@ EMIT LOOP
   LOOP  DROP  ;
```

```
15
\ Reconfiguration clavier : CONFIG.                         15Oct86JaD

CONFIG  (S n --- <nom> ) mot de définition.
            Crée une entrée dans le dictionnaire, et y place le
         texte suivant du flot d'entrée (clavier ou bloc).
            A l'exécution, le mot <nom> empile son adresse,
         et son contenu est emis vers le terminal.
```

```
7
\ Reconfiguration clavier : CLAV-STD.                23Oct86JaD

 28 CONFIG CLAV-STD
0;59;"0 "p&                     0;60;" C0 "p&
0;61;"! "p&                     0;62;" C! "p&
0;63;"OPEN "p&                  0;64;" LIST";13p&
0;65;" EDIT";13p&               0;66;" INDEX";13p&
0;67;" . "p&                    0;68;" .S";13p&
0;71;"";13p&                    0;72;"C-HT";13p&
0;73;"B L";13p&                 0;77;"C-DR";13p&
0;75;"C-GCH";13p&               0;80;"C-BAS";13p&
0;79;"FILE?";13p&               0;81;"N L";13p&
0;82;"' CLAV-ED IS CLAVIER CLAVIER";13p&
0;83;"DARK";13p&
```

```
16
\ Reconfiguration clavier : codes touches fonctions.  18Oct86JaD

        norm.   shift   ctl-    alt-
F1      0;59    0;84    0;94    0;104
F2      0;60    0;85    0;95    0;105
F3      0;61    0;86    0;96    0;106
F4      0;62    0;87    0;97    0;107
F5      0;63    0;88    0;98    0;108
F6      0;64    0;89    0;99    0;109
F7      0;65    0;90    0;100   0;110
F8      0;66    0;91    0;101   0;111
F9      0;67    0;92    0;102   0;112
F10     0;68    0;99    0;103   0;113
```

```
8
\ Reconfiguration clavier : CLAV-ED.                 23Oct86JaD

 17 CONFIG CLAV-ED
0;71;"1 LIST";13p&           0;72;"A B L";13p&
0;73;"B L";13p&              0;77;"A L";13p&
0;75;" L";13p&               0;80;"A N L";13p&
0;79;" CAPACITY 2/ LIST";13p&  0;117;"CAPACITY 1- LIST";13p&
0;81;"N L";13p&              0;83;"WIPE";13p&
0;82;"' CLAV-STD IS CLAVIER CLAVIER";13p&
0;114;""p&                   0;115;""p&
0;116;""p&                   0;118;""p&
0;119;""p&                   0;132;""p&
```

```
17
\ Reconfiguration clavier : codes touches fonctions.  15Oct86JaD

Home        0;71            CTL-Home      0;119
Curs. g     0;75            Curs. dr      0;77
CTL-curs._g 0;115           CTL-curs._dr  0;116
Fin         0;79            CTL-Fin       0;117
Curs._bas   0;80            Curs. haut    0;72
PgUp        0;73            CTL-PgUp      0;132
PgDn        0;81            CTL-PgDn      0;118
Ins         0;82
Annul       0;83
CTL-ImpEc   0;114
TAB         0;15
```

```
+----------------------------------------+
|   IBM Multistation 5550 日本語 A P L    |
|              Version 1.01              |
|  (C) Copyright IBM Corp. 1984, 1985    |
+----------------------------------------+

              Produced by
     IBM Tokyo  Scientific Center
     IBM Madrid Scientific Center

CLEAR WS
      City←'東京　名古屋大阪　長崎　'
      ⍴City
12
      City←4 3⍴City
      City
東京
名古屋
大阪
長崎
      ⌽City
  京東
屋古名
  阪大
  崎長
      ⍉City
東名大長
京古阪崎
  屋
      Yomi←'とうきょうなごや　おおさか'
      Yomi←Yomi,'ながさき　'
      ⍴Yomi
20
      ⌽Yomi
 きさがな かさおお  やごなうょきうと
      Yomi←4 5⍴Yomi
      Yomi
とうきょう
なごや
おおさか
ながさき
      Data←?4 3⍴9999
      Data
 1316 7556 4587
 5328 2190  471
 6788 6793 9346
 3835 5194 8309
      City,'    ¥555.553'⍕Data
東京        ¥1.316      ¥7.556      ¥4.587
名古屋      ¥5.328      ¥2.190        ¥471
大阪        ¥6.788      ¥6.793      ¥9.346
長崎        ¥3.835      ¥5.194      ¥8.309

(Continued)
      City∧.='大阪　'
0 0 1 0
      (City∧.='大阪　')⍳1
3
      Data[(City∧.='大阪　')⍳1;]
6788 6793 9346
      ∇a←Lookup p
[1]   a←Data[(City∧.=p)⍳1;]
[2]   ∇
      Lookup '長崎　'
3835 5194 8309
      +/Lookup '長崎　'
17338
      ⍴AIUEO
55
      ⌽⍉11 5⍴AIUEO
こんるやまはなたさかあ
ゃがれゆみひにちしきい
ゅぎろよむふぬつすくう
ょくわらめへねてせけえ
っげをりもほのとそこお
      AIUEO⍋Yomi
3 1 4 2
      City[AIUEO⍋Yomi;]
大阪
東京
長崎
名古屋
      ∇Report;inx;p;d
[1]   inx←AIUEO⍋Yomi
[2]   p←City[inx;]  ⍝都市名を読みでソート
[3]   d←Data[inx;]  ⍝対応するデータもソート
[4]   ⎕←p,'   ¥555,553'⍕d
[5]   ⎕←''
[6]   ⎕←'合計   ','   ¥555,553'⍕+/[1]d
[7]   ∇
      Report
大阪        ¥6.788      ¥6.793      ¥9.346
東京        ¥1.316      ¥7.556      ¥4.587
長崎        ¥3.835      ¥5.194      ¥8.309
名古屋      ¥5.328      ¥2.190        ¥471

合計       ¥17.267     ¥21.733     ¥22.713
```

JEDI présente en exclusivité le document qui a fait la réputation d'illisibilité de l'APL. Un programmeur bien structuré qui passait par là, a déclaré l'APL "illisible" à tout jamais.

En fait, il y avait de l'APL, mais aussi du japonais. Tous les langages (y compris l'APL et le japonais) ont ceci en commun, c'est que tant qu'on ne les connais pas, on n'y comprend rien.